Safe AI for CPS

(Invited Paper)

Nathan Fulton
Carnegie Mellon University
Computer Science Department
nathanfu@cs.cmu.edu

André Platzer
Carnegie Mellon University
Computer Science Department
aplatzer@cs.cmu.edu

***Abstract*—Autonomous cyber-physical systems – such as self-driving cars and autonomous drones – often leverage artificial intelligence and machine learning algorithms to act well in open environments. Although testing plays an important role in ensuring safety and robustness, modern autonomous systems have grown so complex that achieving safety via testing alone is intractable. Formal verification reduces this testing burden by ruling out large classes of errant behavior at design time. This paper reviews recent work toward developing formal methods for cyber-physical systems that use AI for planning and control by combining the rigor of formal proofs with the flexibility of reinforcement learning.**

I. Introduction

The automotive and aeronautical industries have continually improved the efficiency, safety, comfort and automation of vehicles. Achieving these improvements required substantially increasing the size and complexity of vehicle software. The growing use of software in these safety-critical settings inspired the development of testing, verification, and validation technologies for control systems. Today's self-driving cars, for example, have become so complex that it is completely infeasible to establish their (statistical) safety by testing [1].

Building highly-trustworthy autonomous control systems is particularly difficult because of the interaction between uncertain sensing, discrete computation, physical motion, and real-time interaction with other agents. Verification can help reduce this seemingly intractable burden by ruling out entire classes of errant behavior.

In order to reach the highest mathematical standard for a correct development, developers of safety-critical systems should construct a model of the system under control and then write a formal, computer-checked proof that their control software satisfies key safety properties with respect to the underlying model. For example, a developer might construct a system of differential equations describing how a car drives and then prove that a piece of control software prevents the car from entering an unsafe state. Formal proofs of relevant safety properties ensure that a system is *verifiably safe*.

The demand for formal safety guarantees about control systems led to the development of model checkers and theorem provers for dynamical systems with both discrete and continuous-time dynamics. Hybrid systems provide a fruitful formalism for stating and proving safety properties about traditional control systems that combine discrete computation with continuous control [2], [3], [4], [5], [6], [7].

Autonomous systems, however, increasingly make use of functionality beyond low-level control problems. Software systems not only help control the engine and the brakes, but also make high level decisions about where and how a vehicle should move in the first place. Advanced driver-assist systems are already deployed, and most major automobile manufacturers are experimenting with fully autonomous vehicles. Designers of planes and trains are also deploying partially autonomous vehicles and experimenting with fully autonomous systems. The future of mobility is autonomous.

Autonomous systems make use of artificial intelligence (AI), including reinforcement learning for planning/control and machine learning for perception [8], [9]. Designing safe autonomous systems requires developing formal methods for systems that use reinforcement learning and other optimization techniques for control, but traditional verification approaches do not explain how to obtain safety guarantees for these types of algorithms.

The most fundamental mismatch comes from the following discrepancy. Formal verification benefits from simplicity in order to enable strong and comprehensive predictions about *all* possible behaviors of a system. Artificial intelligence techniques, instead, are lauded for their flexibility in even handling unpredictable situations, which, however, makes it harder to predict the exact behavior of an AI algorithm. Enabling safe advanced autonomy requires the strong predictions that formal methods provide alongside the strong flexibility that the use of AI provides.

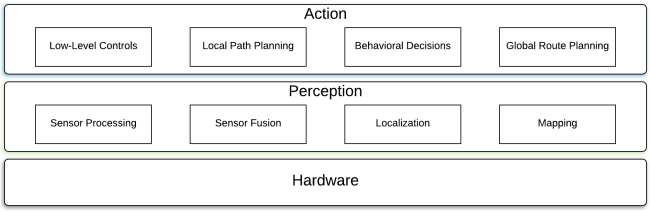

Fig. 1. A high-level overview of an autonomous vehicle's software stack. This diagram is inspired by Paden et al. [8], the Apollo project documentation [10], Pendelton et al [9], and others.

As a prototypical concrete example, Figure 1 illustrates a common decomposition of autonomous systems. The controls & planning layer is composed of a hierarchy of components, each of which solves problems at different levels of abstraction. At the highest level, a global route planner selects a route (e.g., through a road network) from the vehicle's current location to

978-1-5386-8382-8/18/$31.00

its destination.

Executing the global route plan requires recognizing and switching between various contexts (e.g., navigating intersections or changing lanes). The behavioral decision-making layer is responsible for recognizing and switching between these contexts.

This paper reviews our approach toward obtaining formal safety proofs for the planning & controls portion of this software stack. We begin in Section II by explaining how theorem provers establish safety properties about control software for physical processes. Section III explains how low-level control safety properties are lifted in order to provide formal guarantees for reinforcement learning and path planning algorithms. Finally, Section IV shows how safety properties about reinforcement learning algorithms can be used to obtain safety guarantees for behavioral decision making.

Perhaps the most important take-away from this paper is that by starting within the right logical foundations, one can systematically extend verification results about low-level control software up the planning & controls software stack.

After reviewing one holistic approach toward obtaining safe guarantees for planning & controls software, we discuss alternative approaches for each layer of this stack. The paper concludes with a discussion of recent work on formal guarantees for perception and a discussion of future directions toward unifying safety arguments about perception with safety arguments for planning & controls.

II. Verification of Model-Based Controllers

The lowest level of the controls & planning software stack uses (an estimation of) the current state to choose actuator inputs. Low-level controllers are designed with respect to a set of differential equations that model the environment. For over a century, control theorists and roboticists have paid considerable attention to safety, robustness, stability, and optimality of control.

Work toward formalizing the rich theory of control in mechanized, computer readable proofs is relatively new. Our approach is based on *hybrid programs* [4], [6], [11], a programming language that captures the essence of how computation and physics interact in control systems. Hybrid programs are defined by the following grammar, where x ranges over real-valued variables and θ over the terms of real arithmetic:

$$\alpha, \beta \ ::= \ x := \theta \mid x := * \mid \alpha \cup \beta \mid ?\varphi \mid \alpha^* \mid x' = \theta \,\&\, \varphi$$

The first four syntactic forms provide a simple programming language for expressing nondeterministic imperative programs. The assignment program $x := \theta$ assigns to x the value θ. The nondeterministic assignment $x := *$ assigns any arbitrary real number to x. The nondeterministic choice program $\alpha \cup \beta$ executes either α or β. The test program $?\varphi$ evaluates the formula φ and aborts program execution if φ is false. The nondeterministic repetition program α^* repeats α zero or more times. Hybrid programs are hybrid because they combine these discrete programs with continuous programs, i.e., systems of differential equations $x' = \theta$ subject to an evolution domain constraint φ, which is a formula describing the region that the system is not allowed to leave while following this differential equation. More details are in the literature [12].

Example 1 (Safety specification for straight-line car model): The following program models a car moving along a straight line subject to actuator disturbance. The car may choose to accelerate (if safe) or to brake at each time step (at the latest after reaction time T) and the car's physical movement is subjected to an arbitrary bounded disturbance $d_{min} \leq d \leq d_{max}$.

$$\begin{aligned}
\texttt{accel} \equiv\ & ?\Big(p + \frac{(A+d_{max})T^2}{2} + Tv\Big) > \\
& \frac{(v+T(A+d_{max}))^2}{-2(-B+d_{max})}\Big); a := A \\
\texttt{brake} \equiv\ & a := -B \\
\texttt{dist} \equiv\ & d := *;\ ?d_{min} \leq d \leq d_{max} \\
\texttt{plant} \equiv\ & \{p'{=}v,\ v'{=}a+d, t' = 1 \,\&\, v \geq 0 \wedge t \leq T\} \\
& \big(\underbrace{\{\texttt{accel} \cup \texttt{brake}\}}_{\textit{controller}}; \texttt{dist}; t := 0; \texttt{plant}\big)^*
\end{aligned}$$

Differential dynamic logic (dL) [4], [6], [11], [12] is a logic for specifying and proving properties of hybrid programs. Each hybrid program α is associated with modal operators $[\alpha]$ and $\langle\alpha\rangle$, which express state reachability properties of the parametrizing program. The formula $[\alpha]\phi$ states that the formula ϕ is true in all states reachable by the hybrid program α. Similarly, $\langle\alpha\rangle\phi$ expresses that the formula ϕ is true after some execution of α. The dL formulas are generated by the grammar

$$\phi \ ::= \ \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

where θ_i are arithmetic expressions over the reals, ϕ and ψ are formulas, α ranges over hybrid programs, and $\sim$ is a comparison operator $=, \neq, \geq, >, \leq, <$. The quantifiers quantify over the reals. We denote by $\vdash P$ the fact that P has a proof in the dL proof calculus [6], [11], [12].

Example 2 (Safety specification for straight-line car model): The following specification states the car's position (p) must stay behind the position of a static obstacle (o) even when its choices of acceleration are subjected to some bounded disturbance $d_{max} \leq d \leq d_{max}$.

$$\begin{aligned}
\texttt{constBounds} &\equiv A > 0 \wedge B > 0 \wedge d_{min} \leq d \leq d_{max} < B \\
\texttt{brakingDist} &\equiv o - p > \frac{v^2}{-2(-B+d_{max}}) \\
\varphi &\equiv \texttt{constBounds} \wedge \texttt{brakingDist}
\end{aligned}$$

$$\varphi \rightarrow [(\{\texttt{accel} \cup \texttt{brake}\}; \texttt{dist}; t := 0; \texttt{plant})^*] \underbrace{p < o}_{\text{post cond.}}$$

Proving that $\vdash \varphi \rightarrow [\{\texttt{ctrl}; \texttt{ode}\}^*]\psi$ for arbitrary control programs or differential equations is undecidable, meaning that developers must construct these proofs using a combination of manual effort and heuristic automation. The KeYmaera X system is a theorem prover that provides a language called Bellerophon for scripting proofs of dL formulas [13], [14].

The next two sections of this paper show how theorem proving for dℒ provides a foundation for obtaining safety results for cyber-physical systems that use AI algorithms for control. We achieve this goal by lifting safety specifications such as Example 2 to the local path planning and behavioral decision making layers of of the controls software stack.

III. Verified Model-Based Local Motion Planning

Model-based local motion planning takes as input a high-level but local goal – such as navigating a left-hand turn or changing lanes – and produces as output a set of inputs to the low-level control software. Verifiably safe local motion planning can be characterized in terms of either safe planning or safe reinforcement learning. In both cases, the goal is to achieve a high-level but local objective (e.g., stay-in-lane, turn left, etc.) without violating system safety constraints.

A. Verifiably Safe Reinforcement Learning

Reinforcement learning (RL) is one approach toward solving the local motion planning problem [15]. RL algorithms learn to maximize a reward signal by taking actions and observing the effects of these actions. A a reinforcement learning or RL model (S, A, R, E) consists of a finite set S of states, a finite set A of actions, a numerical reward function $R : S \times A \times S \to \mathbb{R}$ for transitioning from $s_i \in S$ via an action $a \in A$ to $s_{i+1} \in S$, and a function $E : S \times A \to S$ characterizing what state the system transitions to when an action $a \in A$ is taken in a given state $s \in S$.

Since RL algorithms need to assume that the response they observed from an action in a state is related to the response they might expect in the future, they assume that any underlying uncertainty is resolved according to the fixed probability distributions of a Markov Decision Process, which provide the mathematical milieu for reinforcement learning. In that case $E(s, a)$ defines, for each state t, the probability of transitioning from s to t after taking action a.

Reinforcement Learning algorithms search for policies[1] that maximize the long-term cumulative value obtained via the reward signal. In this section we abstract away from any particular reinforcement learning algorithm. Our interface to the learning process is just a pair of functions `choose` and `update`. The `update` function selects the next control action to be executed based upon the current high-level goal, and the `update` function changes the algorithm's internal state in response to observed state transitions and rewards.

Unless probabilistic safety answers suffice, the safety analysis is best done using a possibilistic interpretation, where all possible outcomes are considered (by corresponding nondeterministic choices for example). For example, the safety analysis of the obstacle avoidance problem from Example 2 should consider all possible values of the disturbance d, not just the more likely values. There is no safe argument to ignore, say, disturbances that only happen with probability 5%, because they still might happen at some point during an extended drive.

Justified Speculative Control (JSC) [16] is an approach toward verifiably safe reinforcement learning based upon this observation about the relationship between safety for probabilistic systems and reachability for nondeterministic systems. JSC leverages formal proofs to guarantee that a reinforcement learning algorithm avoids unsafe states. JSC uses dℒ safety specifications to obtain safety guarantees for policies obtained via reinforcement learning. We begin by discussing the role of monitoring in safe RL and then introduce the algorithm that uses these monitors to lift control-level safety properties to the planning layer.

[1] mappings from states to actions

B. ModelPlex Monitors

Approaches toward safe control require runtime monitoring; i.e., the ability to check, at runtime, whether or not the current state of the system can be explained by the model of a dℒ formula. The KeYmaera X theorem prover provides a mechanism for translating a dℒ formula of the form $P \to [\alpha^*]Q$ into a formula of real arithmetic, which checks whether the present behavior of a system fits to this model. The resulting arithmetic condition is checked at runtime and is accompanied by a correctness proof. This algorithm, called ModelPlex [17], can be used to extract provably correct monitors that check compliance with the model as well as with the controller. If non-equivalence transformations have been used in the ModelPlex monitor synthesis proofs, the resulting monitor may be conservative, i.e. raise false alarms. But if the monitor formula evaluates to true at runtime, the execution is guaranteed to be safe.

ModelPlex controller monitors are boolean functions that monitor whether or not the controller portion of a hybrid systems model has been violated. The monitor takes two inputs – a "pre" state and a "post" state. If the controller monitor returns true, then there is an execution of the $ctrl$ fragment of the program that maps the previous state to the current state. For example, the controller monitor for Example 2 is:

$$
\begin{aligned}
&\big(\mathbf{a_{post} = A} \land d_{min} \le d_{post} \le d_{max} \land T \ge 0 \land v \ge 0 \land \\
&o - \Big(p + \frac{(A + d_{max})T^2}{2} + Tv\Big) > \frac{(v + T(A + d_{max}))^2}{-2(-B + d_{max})} \\
&\land d_{post} = d \land v_{post} = v \land p_{post} = p \land t_{post} = 0 \land o_{post} = o\big) \\
&\lor \big(\mathbf{a_{post} = -B} \land o - p > \frac{v^2}{-2(-B + d_{max})} \land T \ge 0 \\
&\land d_{min} \le d_{post} \le d_{max} \land v_{post} = v \land p_{post} = p \land o_{post} = o\big)
\end{aligned}
$$

This control monitor is a disjunction between a monitor for the acceleration case `accel` in Example 2 and a monitor for the braking case `brake`. In each case, the monitor checks that control guards are enforced and that variables which should not change remain constant (e.g., $v_{post} = v$ states that the controller should not change the value of the velocity variable).

ModelPlex can also produce full model monitors, which check that the *entire* system model is accurate – including the model of the system's physics – for each round of a control loop. If the full model monitor returns true, then the controller for the system chose a control action that is allowed by the model of the system and also the observed physics of the system correspond to the differential equations describing the

system's physical dynamics.

$$
\begin{aligned}
&o_{post} = o \wedge d_{post} = d \wedge \cdots \wedge \\
&\big(\mathbf{a_{post}} \mathbf{= A} \wedge p_{post} = \frac{(A+d)t^2}{2} + vt + p \wedge \\
&v_{post} = (A+d)t + v \wedge (A+d)t + v \geq 0\big) \quad \vee \\
&\big(\mathbf{a_{post}} \mathbf{= -B} \wedge p_{post} = \frac{(-B+d)t^2}{2} + tv + p \wedge \\
&v_{post} = (-B+d)t + v \wedge (-B+d)t + v \geq 0\big)
\end{aligned}
$$

Portions of the model monitor are elided for concision, but the salient aspects of the monitor for Example 2 are presented above. The monitor checks that constants do not change, and checks that each variable comports with the solution to the differential equations. By relating a dℒ model to runtime monitoring constraints, we can leverage formal guarantees to obtain safety results for a reinforcement learning algorithm.

C. The Justified Speculative Control Algorithm

The JSC algorithm explains how to leverage ModelPlex monitors to lift control-level safety guarantees to the planning layer. JSC achieves this goal by modifying a generic reinforcement learning algorithm with runtime monitors that restrict the set of actions available to the agent. A formal proof connecting these runtime monitors to an original dℒ model ensures that the learning agent only takes actions that will be safe in the underlying environment model.

The Justified Speculative Control algorithm takes as input an initial state, an RL model, a reinforcement learning algorithm characterized in terms of its `choose` and `update` functions, a characterization `done` of terminal states, and runtime monitors (a control monitor *CM* and a model monitor *MM*). When the environment is accurately modeled (i.e., *MM* returns true) the control monitor *CM* is used to constrain the learning algorithm's choice of available actions to a known safe subset of available actions. However, when the environment is not accurately modeled, the RL agent's choice of actions is not constrained, because it evolves outside known safe actions.

JSC uses runtime monitors to constrain the set of available actions. Lines 5 and 6 specify that if the environmental model is accurate according to a runtime monitor *MM* for the model, then the learning agent may only select actions that the controller runtime monitor *CM* designates as proven safe.

The JSC algorithm guarantees that if the differential equations describing the environment E are accurate, then the learned controller will never enter a state where the post-condition of the original dℒ specification is violated. This Safe Learning theorem [16, Thm. 1] also implies that policies extracted from the reinforcement learning algorithm are safe.

Justified Speculative Control Learning

```
1 JSC(init, (S,A,R,E), choose, update, done, CM, MM) {
2   prev := curr := init;
3   a0   := NOP;
4   while (!done(curr)) {
5     if (MM(prev, a0, curr))
6       u := choose({a ∈ A | CM(a,curr)});
7     else
8       u := choose(A);
9     prev := curr;
10    curr := E(u, prev);
11    update(prev, u, curr);
12  }
13 }
```

D. Off-Model Safety

JSC guarantees, if we use runtime monitors that provably correspond to a proved dℒ formula init $\rightarrow [\{ctrl; plant\}^*]$safe in order to sandbox a reinforcement learning algorithm, then the safety constraint (safe) is maintained for any learned policy *as long as the differential equations plant are accurate*.

JSC goes beyond mere sandboxing in a best-effort attempt to deal with the problem of guaranteeing safety even when modeling constraints are violated. Lines 5–8 of Listing III-C allow *any* action whenever the model monitor evaluates to false. A refined version of the algorithm [16] refines `choose` to depend on whether *MM* evaluated to true in the previous time step. When the model monitor *MM* is false, we introduce a new reward signal that leverages insights from our formal specification in order to direct reinforcement learning. We use a quantified version of the original model monitor *MM*, effectively rewarding the system for minimizing the error between the model's prediction of what should have happened and the sensor's observations of what actually happened. This was experimentally observed to have the effect of driving the agent back into known safe portions of the state space quickly.

Local path planning problems are often characterized in terms of a Markov Decision Process whose states and actions correspond to the states and actions used for low-level verified control software. When this is the case, the JSC algorithm transfers safety results to learned policies whenever the model is accurate, and extensions to JSC can leverage information from the verified model to direct path planning even when the environment was not accurately modeled[2].

IV. Model-Based Behavioral Decision Making

JSC uses verified monitors to extend verification results about low-level control software to learned behaviors for achieving high-level but local goals, such as navigating an intersection or optimally following a leader car. The next level up in the autonomous vehicle software stack tackles the problem of choosing and switching between multiple available behavioral models.

Paden et al. illustrate this layer of the stack in Figure 2 [8]. As this graphic demonstrates, the task of switching between these tasks depends on a combination of the global route planning goals of the system and the behavior of other agents in the system. For example, lane switching might be triggered by the behavior of drivers in the vehicle's current lane or by the global route planner prescribing a left turn.

Behavioral decision making includes two subtasks: identifying which behavioral mode the system should be operating in and then controlling safely within that mode. We achieve these two goals by adding an outer loop that performs falsification over a set of feasible models and then performs JSC with the currently feasible models. We call this combination of runtime falsification with JSC model update learning.

[2]although safety guarantees are, obviously, not retained unless model deviations are bounded [17]

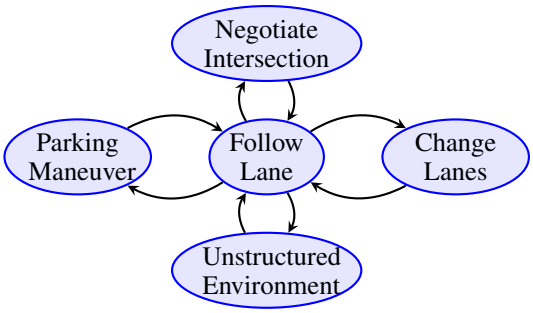

Fig. 2. The Behavioral Decision Making problem [8]

We define our model update learning algorithm (μ-learning) in terms of *feasible (i.e., currently unfalsified) monitored models*. A modeled monitor is simply a model together with its control monitor and model monitor.

Definition 1: A **monitored model** is a tuple (m, CM, MM) where:

- m is a (proved) dL formula of the form `init` $\rightarrow$ $[\{$`ctrl;plant`$\}^*]$`safe` where `ctrl` is a loop-free discrete program and the entire formula m contains exactly one modality.
- The formulas *CM* and *MM* are the control monitor and model monitor corresponding to m.

The μ-learning algorithm takes as input an environment `env`, discrete action space `A`, an initial state `init`, `choose` and `update` functions as in JSC, and a finite set of monitored models `M` each with a method `models` which implements the model monitor and `safe` which implements the controller monitor.

The key property of this algorithm is that it always enforces the runtime controller monitors for all models that have not been ruled out by the observed data. This, combined with the assumption that there exists a distinguished model `m`* $\in$ `M` that accurately models `env`, ensures that the safety properties of the JSC algorithm extend to the behavioral layer whenever selecting a single behavioral model from a set of available models.

Listing 1. Basic μ-learning

```
1 mulearn(init, env, choose, update, done, M) {
2   prev := curr := init;
3   act  := None;
4   while (!done(curr)) {
5     if (act != None) {
6       M := {m ∈ M : m.models(prev,act,post)};
7     }
8     avail := {a ∈ A : ∀ m ∈ M. m.safe(a)};
9     act := choose(avail);
10    prev := curr;
11    curr, reward := env(curr, act);
12    update(prev, act);
13  }
14 }
```

a) Active Experimentation: One downside of μ-learning is the conjunctive constraint on lines 6 and 7. This constraint is often restrictive because we may only take actions that are safe for every model. However, if each model additionally has a prediction method `m.predict`, then μ-learning can be extended to always choose actions that are *active experiments*, in the sense that they eliminate at least one of the currently feasible models. A perhaps surprising fact about active experimentation is that it does not always guarantee convergence to an optimal set of models.

The μ-learning algorithm extends formal safety guarantees from the local planning layer to the behavioral layer by combining verified reinforcement learning with online falsification to choose among a set of possible behavioral models.

A. Off-Model Decision Making

Standard μ-learning assumes that a distinguished accurate model `m`* already is in the monitored models M. This assumption may be relaxed in several ways.

First, we can introduce systematic updates to the environmental model (i.e., the ODEs or the non-control portion of the discrete dynamics) with corresponding proof-preserving updates to the control model. We call this type of update a verification-preserving model update.

Definition 2 (VPMU): A *verification-preserving model update* (VPMU) is a pair of mappings `modelUpdate` and `proofUpdate` which take as input an initial dL formula φ with an associated Bellerophon proof `e` of φ, and produce as output a new dL formula `modelUpdate`(φ) and a proof `proofUpdate(e)` of `modelUpdate`(φ).

VPMUs preserve verification results while also expanding the set of available models beyond what the system designers initially anticipate. This combination of features is attractive – the system designer can carefully state safety specifications and some ways that the system's behavior might change over time, and then iteratively apply all available VPMUs to all available models out to some finite horizon. This approach increases the class of behaviors under which the system can act safely. Formally, our assumption that `m`$^* \in M$ relaxes to an assumption that there is sequence of VPMUs $v_0 \dots v_n$ and an initial model $m_0 \in M$ such that the accurate model `m`* can be obtained from m_0 by this sequence of VPMUs, i.e., $v_n(\cdots(v_0(m_0))\cdots) = m^*$.

A second and much broader way of relaxing the `m`* assumption is to consider model updates that are *not* verification preserving. This approach is analogous to off-model learning in JSC. Once we realize the there is no accurate verified behavioral model (i.e., M is empty), we can then consider whether there is any model – verified or not – that might give us reasonable behavior despite not having a formal proof. Just as in JSC, switching to an unverified model should only happen as a last resort when there is no available verified model that accurately characterizes observed reality.

V. Alternative Approaches Toward Safe AI for Planning & Control

JSC and μ-learning provide an approach toward obtaining verifiably safe AI for planning & control. These approaches leverage dL to characterize safety properties about random systems in terms of reachability constraints on nondeterministic systems. The model monitor synthesis algorithms implemented

in KeYmaera X are enabling technology for both of these safe AI algorithms.

There are several alternative approaches toward hybrid systems verification, including δ-decision procedures [18], [7] and model checking of hybrid automata [5]. They can justify the safety of a model but lack dL's ModelPlex ability for synthesizing runtime monitors that are accompanied by a priori correctness proofs. The combination of both, safety proof and runtime monitor proof, is used to build safe AI for CPS.

Alshiekh et al. and Hasanbeig et al. each introduce approaches toward safe RL based upon temporal logics [19], [20]. Unlike JSC, these approaches do not use a logic capable of expressing hybrid system dynamics, and, thus, are missing one important ingredient for CPS control. Alshiekh et al. use deterministic safety work automata to abstract over the MDP's dynamics and Hasanbeig et al. construct Limit Deterministic Büchi Automata from LTL specifications. The use of LTL and non-hybrid automata limits the applicability of these approaches in cyber-physical systems, but does succinctly capture many constraints on discrete or discretized planning and optimization problems. The combination of online learning with safety has also been proposed for Hamilton-Jacobi equation solving [21], [22], experimentally demonstrating the potential of mixing learning and verification for quadrotor flight.

Although verifiably safe AI is a new research field, existing approaches suggest three important criteria in the context of safety-critical controls & planning software.

- Approaches toward safe AI for CPS should provide simple and familiar representations of environmental assumptions (preferably ODEs and imperative programs, the *lingua franca* of classical control). Environmental models are the key to lifting safety guarantees about local planning to safety guarantees about behavioral decision making. Explicit environmental models are also the only lifeline available when model deviation occurs. Detecting model deviation and finding ways to correct for deviation is impossible unless an explicit environmental model is available.
- Environmental assumptions should be easy to compose into hierarchical decision making frameworks.
- Approaches toward safe AI for CPS should provide formal proofs, not just formal specs. Therefore, approaches must provide a way of relating monitoring conditions, safety specifications, and environmental assumptions via formal proof. This should be possible even for the nonlinear systems important in applications.

VI. Verified Perception

Applying formal methods to perception is inherently difficult because formally specifying correctness for perception algorithms remains an open problem.

The predictor/verifier framework of Dvijotham et al. [23] introduces an approach toward obtaining proofs that neural networks are robust to adversarial perturbations. The VeriVis methodology of Pei et al. characterizes safety in terms of explicit attacker capabilities [24].

One obvious next step toward safe AI for CPS is to obtain end-to-end safety guarantees by composing approaches toward safe perception with approaches toward safe control. To facilitate this, approaches toward safe AI for CPS should build on a multi-dynamical and extensible logical foundations so that safety guarantees about perception can be incorporated into safety arguments for controls and planning.

VII. Future Work

Existing approaches toward verification of model-based planning/optimization incorporate formal guarantees into the learning/optimization process by synthesizing monitors from some combination of explicit safety constraints on actions, environmental models, and synthesized monitors. We call this family of approaches *formally constrained learning/optimization in model space* because the constraints are stated in a formal language and safety guarantees apply only as long as the environmental model is accurate.

Future work on safe AI for planning and control will focus on moving beyond these table stakes by: 1) providing safety and optimality guarantees for continuous state and action spaces; 2) providing safety and optimality guarantees outside of model space; i.e., when the environment model is not accurate or unavailable (this is discussed at greater length in Section IV-A); 3) extending approaches that provide *unverified* formal constraints with formal mechanized proofs that action constraints are safe with respect to the environmental model; and 4) incorporating formal guarantees for learning into verified pipelines such as VeriPhy [25], which maps high-level hybrid systems models down to provably correct machine code implementations of model monitoring constraints. This could be done by leveraging existing mechanized proofs about MDPs [26].

VIII. Conclusion

This paper demonstrates one approach toward obtaining safety guarantees for the control portion of an AI-enabled autonomous vehicle using a combination of hybrid systems theorem proving, monitor synthesis, monitor-constrained reinforcement learning, and runtime falsification. This particular approach uses the KeYmaera X theorem prover's implementation of differential dynamic logic, but many of our ideas and algorithms generalize to any other future technology capable of verifying and generating runtime monitors for hybrid dynamical systems with correctness proofs. The requisite capabilities in dL stem from its ability to mix $[\alpha]$ and $\langle\alpha\rangle$ modalities for all and some runs of a hybrid system α.

The future of Safe AI research will focus on expanding existing methodologies to obtain guarantees for off-model learning and on combining formal safety specifications for perception with formal safety proofs for planning and control.

Acknowledgment

We thank the ITC conference for the kind invitation to present this work. This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) under grant number FA8750-18-C-0092, and by the Future of Life Institute (futureoflife.org) FLI-RFP-AI1 program, grant #2015-143867.

REFERENCES

[1] N. Kalra and S. M. Paddock, "Driving to safety – how many miles of driving would it take to demonstrate autonomous vehicle reliability?" RAND Corporation, Tech. Rep., 2016.

[2] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., vol. 736. Berlin: Springer, 1992, pp. 209–229.

[3] A. Nerode and W. Kohn, "Models for hybrid systems: Automata, topologies, controllability, observability," in *Hybrid Systems*, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., vol. 736. Berlin: Springer, 1992, pp. 317–356.

[4] A. Platzer, "Differential dynamic logic for hybrid systems." *J. Autom. Reas.*, vol. 41, no. 2, pp. 143–189, 2008.

[5] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *23rd CAV, 2011*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 379–395.

[6] A. Platzer, "Logics of dynamical systems," in *LICS*. IEEE, 2012, pp. 13–24.

[7] S. Kong, S. Gao, W. Chen, and E. M. Clarke, "dReach: δ-reachability analysis for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, ser. LNCS, C. Baier and C. Tinelli, Eds., vol. 9035. Springer, 2015, pp. 200–205.

[8] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[9] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[10] The Apollo Project, "ApolloAuto: An open autonomous driving platform," 2018. [Online]. Available: https://github.com/ApolloAuto/apollo

[11] A. Platzer, "A complete uniform substitution calculus for differential dynamic logic," *J. Autom. Reas.*, vol. 59, no. 2, pp. 219–265, 2017.

[12] ——, *Logical Foundations of Cyber-Physical Systems*. Switzerland: Springer, 2018.

[13] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *CADE*, ser. LNCS, A. P. Felty and A. Middeldorp, Eds., vol. 9195. Springer, 2015, pp. 527–538.

[14] N. Fulton, S. Mitsch, B. Bohrer, and A. Platzer, "Bellerophon: Tactical theorem proving for hybrid systems," in *ITP*, ser. LNCS, M. Ayala-Rincón and C. A. Muñoz, Eds., vol. 10499. Springer, 2017, pp. 207–224.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[16] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, February 2-7, 2018, New Orleans, Louisiana, USA.*, S. McIlraith and K. Weinberger, Eds. AAAI Press, 2018, pp. 6485–6492.

[17] S. Mitsch and A. Platzer, "ModelPlex: Verified runtime validation of verified cyber-physical system models," *Form. Methods Syst. Des.*, vol. 49, no. 1, pp. 33–74, 2016, special issue of selected papers from RV'14.

[18] A. Eggers, M. Fränzle, and C. Herde, "SAT modulo ODE: A direct SAT approach to hybrid systems," in *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings*, S. D. Cha, J. Choi, M. Kim, I. Lee, and M. Viswanathan, Eds., vol. 5311. Berlin: Springer, 2008, pp. 171–185.

[19] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018.

[20] M. Hasanbeig, A. Abate, and D. Kroening, "Logically-correct reinforcement learning," *CoRR*, vol. abs/1801.08099, 2018.

[21] J. H. Gillula and C. J. Tomlin, "Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor," in *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE, 2012, pp. 2723–2730.

[22] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. H. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *CoRR*, vol. abs/1705.01292, 2017.

[23] K. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelovic, B. O'Donoghue, J. Uesato, and P. Kohli, "Training verified learners with learned verifiers," *CoRR*, vol. abs/1805.10265, 2018.

[24] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: The case of computer vision systems," *CoRR*, vol. abs/1712.01785, 2017.

[25] B. Bohrer, Y. K. Tan, S. Mitsch, M. O. Myreen, and A. Platzer, "VeriPhy: Verified controller executables from verified cyber-physical system models," in *PLDI*, D. Grossman, Ed. ACM, 2018, pp. 617–630.

[26] J. Hölzl, "Markov chains and Markov decision processes in Isabelle/HOL," *Journal of Automated Reasoning*, vol. 59, no. 3, pp. 345–387, 2017.

[27] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., *Hybrid Systems*, vol. 736. Berlin: Springer, 1993.